66 Aut Comp Conf

Generating And Debugging Programs At Remote Consoles Of The PDP-6 Time Sharing System

By PETER V. WATT,
Digital Equipment Australia Pty. Ltd.,
89 Berry St., North Sydney.

SUMMARY

It is necessary to provide the user at a remote console of a time sharing computer system with some means of examining and controlling the programs he wishes to test. As this control must be effected by a program, it is economically possible to provide a very sophisticated debugging tool which operates in the symbols of the user's source language. DDT, the Dynamic Debugging Technique provided at consoles of the PDP-6 time sharing system, is a powerful debugging aid; its facilities are described in this paper following a summary of relevant aspects of the PDP-6 time sharing system.

INTRODUCTION

In many computer installations the actual time taken to get new jobs running is of vital concern. In this sense "time taken" refers to the actual number of hours which elapse from the definition of a problem to its successful evaluation, not the number of man-hours, or computer-hours, or number of computer runs, or other of the multitude of yardsticks used to measure job complexity, programmer ability, or computer performance. Computer bureaux, Defence installations and University computing centres — indeed, any installations which have a heavy development load — are vitally interested in reducing the time taken to develop programs.

It is not seriously disputed now that close contact with a computer while debugging decreases the amount of real time required to get a new program going. In the past, however, it has not been economically possible to allow programmers close contact — they just make it too difficult to schedule the normal work load of the computer centre. One of the major contributions of time sharing is that it has made on-line reaction between programmer and program feasible. Indeed, the most significant advantage of a multi-access time sharing system as opposed to a more conventional multi-programming system may be this ability to generate programs quickly through on-line reaction between programmer and computer.

Scheduling Algorithms in Single Access and Multiple Access Time Sharing Systems

A single access system is usually scheduled to run jobs in a strict priority sequence. The highest priority job runs until it becomes I/O bound, at which stage the next highest priority job runs until the first job is no longer I/O bound, or until this job itself becomes I/O bound. Priorities are allocated externally (usually by the operator), and the highest priority job runs through to completion very nearly as quickly as it would if it were not in a time sharing system.

In a multi-user system the requirement is to minimize the maximum delay to any one user: to effect this a "round-robin" scheduling system is used. Normally, when one job finishes computing the next is examined. However, jobs which have just made use of input/output equipment are given priority over straight computing jobs. This is based on the assumption that this job is likely to request additional input/output, and represents an attempt to minimize input/output delays.

The scheduler of the PDP-6 system is based on a clock which "ticks" at mains frequency — i.e., once every 20 milliseconds. A job is usually allowed a quantum of 15 "ticks" and can get this quantum as often as computing time is available. If the job requests input/output and cannot continue computing it will be called an "I/O job" and will receive a special quantum of 4 ticks next time it can be scheduled, irrespective of the round-robin arrangement. The job continues to be an I/O job until it exhausts its quantum without calling for I/O.

Remote Console Operation

When a remote console is first turned on it is said to be in Monitor Command Mode. This is a state in which the monitor will try to translate any information typed on the console as though it were a command. The commands currently implemented for the PDP-6 system are detailed in Appendix I. Appendix II lists the special teletype characters used and their functions.

From the console the user may initiate his job, ask for core space, and get the programs he wants from DECtape (or Magnetic tape, drum or disc where these are installed). A typical method of generating a new program would be for the user to ask for the DECtape editor (which requires 1K of core) and to type his program to the editor, which will store it on DECtape. Appendix IV illustrates this type of console conversation. Alternatively, the user may use his remote console off-line to prepare paper tape, or even have his program punched on cards if he so wishes.

Once the program exists in a form in which the System can read it, the user would ask for the Assembler, and assemble it into a relocateable binary file on another DECtape. The Loader would load this binary file into the user's core area, linking his various binary files as necessary. At this stage the user can commence debugging — after saving his core image on tape so that he can restart after catastrophic failures in his program.

In debugging, he talks to a program called DDT. This program is described in some detail in the following pages.

Dynamic Debugging Technique (DDT)

DDT is a user program written to facilitate the debugging of other programs. It allows interaction between the programmer at a remote station and the computer, and provides a flexibility and sophistication not seen at most computer consoles.

The fundamental requirement for debugging a program is to be able to examine and modify selected locations in the program area. At a computer console, these functions are usually available by setting switches to the selected address, pressing a button to examine the contents of this word, setting up switches to the desired (octal) contents, and depressing a second button to deposit the desired contents into the selected address.

The DDT equivalent of this operation is to have the user type the address of the word he wishes to examine, followed by a "slash". DDT will respond by typing the contents of that location, followed by a few spaces. The user can then type in a new value, which will be deposited as the new contents of the selected location. For example:—

157/ 367440000153 365440000153

(underlined characters are typed by the user) would cause DDT to change the contents of location 157 from 367440000153 to 365440000153.

The fact that there is a program assisting the debugging operation (rather than computer hardware) allows some significant improvements in this two-way "conversation". These include the use of the user's source-language symbols, a variety of radices for input and output of information, multiple and conditional breakpoint facilities, and searching facilities.

i. *Source Language Symbols*

The symbols defined in a user's program are carried with the binary program created at assembly time. When

program is loaded by the Linking Loader the user may specifically request symbols to be loaded; if so, symbols are stored in core together with the binary object code.

As loading proceeds, the Linking Loader makes the necessary linkages between out-of-program references; it also leaves the symbols in a form accessible to DDT. Thus while the previous example of the use of DDT is quite correct, it would be more normal for the conversation to read as follows:—

READ+3/

SOJG COUNT, RCHAR SOJGE COUNT, RCHAR

(Appendix III lists the program to which these examples relate.)

It is immediately obvious that this ability to work in source symbols implies a further two functions of DDT — those of assembly and dis-assembly. It is, in fact, possible to sit down at a remote console and, using DDT as an assembler, compose a program. This use is rather frowned upon.

ii. *Choice of Radix.*

The radix of constants typed out by DDT can be set to any value greater than one. Most frequently used radices are 8, 10 and 2 in that order. For input of numbers to DDT, the simple convention that if a number has no decimal point it is octal; if it has a decimal point but no digits after the decimal point it is decimal; and if it has digits to the right of a decimal point it is a floating point number is adopted. Thus radix conversions and conversions from fixed to floating point are easily effected. For example:—

153.=231	Convert from decimal to octal.
12$$R	Set radix to 12 octal (or 10.$$R)
153.=153.	
231=153.	
176=126.	
3.1415927 = 202622077327	(DDT types the octal number which represents the floating point value of π.)
TABLE+5/	Examine TABLE+5.

MOVEM 14. @77327(2)

$$F		Change to floating point
TABLE+5/	3.1415927	output and look again.

iii. *Break Point Facilities.*

DDT allows the programmer to use up to eight break points in the program to be debugged. The basic operation of a break point is, of course, to return control to DDT when the user's program reaches a certain point.

To insert a break point, a command would be typed to DDT:—

READ+2$B	Insert a break point at location READ+2.

When the subject program is started, it will run until it comes to the instruction at READ+2 (which is now a jump to DDT). The conversation might appear:—

READ+2$B	Insert a break point at location READ+2.
START$G	Go to the location called START.
$1B>>READ+2	The program has run up to (but not including) the instruction at READ+2, and has encountered the first break point.
TAC/ 170	Examine TAC.
$$T TAC/ X	Change radix to ASC11 — re-examine TAC.

It is normal to cause the contents of an interesting register to be typed automatically. The conversation proceeds:—

READ+2(TAC)$B	Set a break point at READ+2. When it is encountered, type out TAC in the current radix.
$P	Proceed from the last break point.
$1B >> READ+2 TAC/ 0	Break point number 1 encountered: TAC contains "0".
98.$P	Proceed another 98 times.
$1B>>READ+2 TAC/ Z	Break point number 1 encountered for the 98th time — TAC contains "Z".

With each of the eight possible break points is associated a "Proceed Counter", which will be decremented to zero before a break occurs, and a conditional test which must succeed (or must fail) before the proceed counter is decremented. Thus we could instruct DDT to set a break point which would:—

1. Break on every encounter.
2. Allow the program to run through the break point a specified number of times, and then break.
3. Break on the first ocurrence of some condition, or after a specified number of times through the break point, whichever event comes first.
4. Break on the occurrence of some set of conditions.

The condition tested for can be made as complex as required by replacing the conditional instruction by a jump to a subroutine. Thus the conversation might proceed:—

$1B+1!	JSR	END	
END!	0		
END+1!	AOS	END	
END+2!	CAIN	TAC, 4	MACRO-6 Assembly language subroutine to test if the character is an EOT or an EOM, force a break.
END+3!	JRST	@END	
END+4!	CAIE	TAC, 3	
END+5!	AOS	END	
END+6!	JRST	@END	

This would force a break only if either an "End of Transmission" character (ASCII code 004) or an "End of Message" character (ASCII code 003) was seen.

Thus break point logic can be used to test almost any condition in the program — even to the extent of stopping when a particular instruction is overwritten! The conditional break point facilities provide a powerful debugging aid which helps the programmer to isolate complex logical bugs more quickly than almost any other method.

iv. *Search Facilities.*

Three types of searches may be carried out by DDT — Word searches, Not-word searches, and Effective Address searches. The two former searches allow sections of core to be searched for the occurrence or non-occurrence of selected values. Continuing with the demonstration problem, we might type (after having read a few characters):—

CHARCO<CHARCO+127.>1$W		Search the area between CHARCO and CHARCO+127.: print out the locations which contain the value 1.
CHARCO+3/	1	
CHARCO+12/	1	
CHARCO+15/	1	
CHARCO<CHARCO+127.>0$N		Search the table and print all locations which are not zero.
CHARCO+3/	1	
CHARCO+12/	3	
CHARCO+15/	3	
CHARCO+30/	1	
CHARCO+41/	2	
CHARCO+42/	1	

Each of these two searches operates in conjunction with a mask. The locations are accessed, logically "Anded" with the mask, and then tested for equality or non-equality with the selected value. Thus the conversation might continue:—

$M/	-1	750000000000 Set mask to this value.
START<IBUF-1>CALL$W		Search for all instructions with 0 in first digit and 4 or 6 in second digit.

```
START/      CALL RESET
START+4/    INIT PTR, 0
START+7/    CALL EXIT
START+11/   INBUF PTR, 2
READ/       INPUT PTR, 0
READ+4/     CALL EXIT
```

This is a list of all instructions in the program which cause a trap to the monitor, to request some monitor activity for the user program.

The third search is one for instructions which "point to" a given location. To demonstrate the use of this, assume that the instruction AOS CHARCOUNT(TAC) at READ+2 had inadvertently been written as AOS CHARCOUNT(TAC1). On running the program we find that the instruction RCHAR has been modified, which is not intended. To help to find why, we might type:—

START<CHARCO+127.>RCHAR$E — Search for effective addresses equal to RCHAR. DDT would respond with:

```
READ+2/    AOS CHARCO (TAC1)
READ+3/    SOJG COUNT, RCHAR
```

This would tell us that instructions at READ+2 and READ+3 are "pointing at" RCHAR (DDT will follow indirect and index chains to 64 deep in calculating the effective address); from this information it would be obvious that TAC1 contains the wrong information, or that we referred to the wrong index register.

DDT as a Debugging Tool for use with Fortran Programs

While DDT is intended primarily as an assembly language tool, it can be used to advantage with Fortran programs. The symbols and statement numbers used in Fortran programs are available to DDT, and break point logic is also useful. (Examination of program locations will be meaningful to the Fortran programmer only if he has requested a listing of his compiled program.)

Consider the Fortran segment:—

```
10    DO 12 I=1, 100.
11    A=A+ATAB(I).
12    ASQ=ASQ+ATAB(I)*ATAB(I).
```

At run time the programmer is able to insert a break point (%12$B will insert a break point before statement 12) and to examine registers in exactly the same way as with an assembly language program. The fact that programs written in Fortran tend to be logically straightforward means that it is more common for Fortran users to try their programs without DDT, and to use DDT only if mistakes are not obvious in the source language.

APPENDIX I

TABLE OF MONITOR COMMANDS

Name	*Function*	*Response*	*Antonym*
IJOB	Initiate a JOB	JOB N INITIALIZED	KJOB
KJOB	Kill this job (i.e., release all core and I/O devices)	LF	IJOB
PJOB	Print the user's job number	1, 2, 3, ... 9 or 10	—
CORE N	Allocates N*1024 words of core store for this job	M FREE BLOCKS LEFT, NONE ASIGNED or LF	—
GET	Get a program from a DECtape, Mag. tape, disc or drum such as: GET SYS FORTRAN (get Fortran compiler from System DECtape); GET DTA5 ACCDEM (get a file called ACCDEM from DECtape number 5)	JOB SETUP	SAVE
SAVE	Save user's core image as a file on DECtape, Mag. tape, disc or drum. E.g., SAVE DTA5 A3 would save user's core image on DECtape 5	JOB SAVED	GET
START	Start to obey the program in the user's core at the start address stored with the program.	LF	↑C
START N	Start to obey the user's program at (octal) location N	LF	↑C
START C	Start the user's program, but keep the teletype in Monitor Command Mode (so that the user can issue further commands to the monitor)	LF	↑C
STARTC N	Start the user's program at location N, with console in Monitor Command Mode	LF	↑C
CONT, CONTC	Continue job from where it left off (presumably as a result of user typing ↑C)	LF, or CAN'T CONTINUE (if program stopped because of errors)	↑C
DDT	Start obeying user's copy of DDT (debugging program)	LF, or NO DDT (if user didn't request DDT to be loaded)	—

Name	*Function*	*Response*	*Antonym*
ASSIGN	Assign the device requested to the user (so that no other user can use it until this user DEASSIGNs it, or kills his job), e.g., ASSIGN DTA5 or ASSIGN DTA5:IN. This would allow the user to call device DTA5 "IN" — both in monitor commands and in his program. ASSIGN PTR:CDR would allow a program written to take input from card reader, to take input from paper tape	DEVICE XXX ASSIGNED or NO SUCH DEVICE or ALREADY ASSIGNED TO JOBN	DEASSIGN
DEASSIGN	Deassign the device requested	LF or DEVICE XXX WASN'T ASSIGNED	ASSIGN
DETACH	Detach the teletype from this job (to allow the user to control more than one job from his teletype). He could then initiate another job, request core, and run a second (or nth) job	LF	ATTACH
ATTACH N	Attach this teletype to job N (to allow the user to assert control of a job initiated at another teletype, or to regain control over a job from which he has DETACH'ed)	LF, or DEVICE TTYN ALREADY ATTACHED or JOB N NEVER WAS INITIALIZED	DETACH
OPR	Transfer all characters until C to operator's console	LF or ENGAGED	—
WRU, or special WRU character	Provides answer-back facility for TWX and TLX networks	PDP-6 UNIWEST	—
DIGITAL	Serves no purpose	HELLO	

APPENDIX II

TABLE OF SPECIAL CHARACTERS AND THEIR USES

Character Input	*ASCII Value*	*Function*	*Response from System*
CARRIAGE RETURN	15	Terminate input (monitor command mode). May terminate input in user input	LF
LINE FEED	12	Ignored	
RUBOUT	177	Delete preceding character (successively)	
(CONTROL)C	03	Terminate user's job as soon as possible; return to monitor command mode	↑C
(CONTROL)O	17	Kill current output buffer	↑O
(CONTROL)U	25	Kill current input buffer	U
(CONTROL)P	20	Change setting of this teletype to (or from) automatic tab, vertical tab, and form feed	—
(CONTROL)Z	32	End of text, or end of data	—

Note: (CONTROL) is a special key on the teletype, used in conjunction with another key. Thus "(CONTROL)P" indicates the "control" key was held down while "P" was depressed.

APPENDIX III

DEMONSTRATION PROGRAM

```
CHARACTER COUNT                                          PAGE 1
     ACC DEMONSTRATION PROGRAM

                                    ;
                                    PTR=1
                                    A=0
                                    TAC=10
                                    TAC1=12
                                    NOCHAR=^D5000
                                    COUNT=11
                                    ;
                                    START:    CALL RESET          ;ALL PROGRAMS START THIS WAY
                                              SETZM CHARCOUNT     ;CLEAR FIRST WORD OF TABLE
                                              MOVE TAC,[XWD CHARCOUNT,CHARCOUNT+1]    ;SET UP TRANSFER POI
NTER
                                              BLT TAC,CHARCOUNT+^D127       ;ZEROISE COUNT AREA
                                    ;
                                              INIT PTR,A          ;INITIATE A DEVICE ON CHANNEL 1
                                              SIXBIT /PTR/        ;DEVICE IS PAPER TAPE READER
                                              EXP IBUF  ;WITH BUFFER HEADER AT "IBUF"
                                              CALL EXIT ;EXIT IF PIP NOT AVAILABLE
                                    ;
                                              MOVEI COUNT,NOCHAR  ;SET UP CHARACTER COUNT
                                    ;
                                              INBUF PTR,2         ;SET UP A 2-BUFFER RING
                                              SKIPA               ;AND FALL INTO "INPUT"
                                    ;
                                    NCHAR:    SOSG IBFCNT         ;ANY CHARACTERS LEFT IN THIS BUFFER?
                                    READ:     INPUT PTR,0         ;NO - GET ANOTHER BUFFER FULL
                                              ILDB TAC,IBFPTR     ;YES - GET THE NEXT CHARACTER
                                    ;
                                              AOS CHARCOUNT(TAC)  ;INCREMENT THE APPROPRIATE SLOT IN THE T
HE TABLE
                                              SOJG COUNT,NCHAR    ;AND GO BACK FOR MORE CHARACTERS
                                              CALL EXIT ;UNLESS WE HAVE ALREADY READ 5000
                                    ;
                                    RESET:    SIXBIT /RESET/
                                    EXIT:     SIXBIT /EXIT/
                                    ;
                                    IBUF:     0
                                    IBFPTR:   0
                                    IBFCNT:   0
                                    ;
                                    CHARCOUNT:          BLOCK ^D128         ;TABLE FOR CHARACTER COUNTS
```

```
CHARACTER COUNT                                          PAGE 2
     ACC DEMONSTRATION PROGRAM

                                              END START
```

```
CHARACTER COUNT
     SYMBOL TABLE
```

APPENDIX IV

TYPICAL REMOTE CONSOLE CONVERSATION

```
;TYPICAL SYSTEM CONVERSATION AT A REMOTE CONSOLE:

IJOB
JOB 2 INITIALIZED

CORE 2

ASSIGN DTA
DEVICE DTA6 ASSIGNED

OPR
PLEASE LOAD TAPE 53 ON DRIVE 6 AND ENABLE WRITE SWITCH
GA

OK DONE GA
TKS BYE
^C
GET SYS PIP2
JOB SETUP

START

TTY:=DTA6:/L
ACCDEM.TXT      12/22/65
ACCDEM.DMP      12/23/65

562. FREE BLOCKS LEFT

^C
CORE 1
GET SYS EDITOR
JOB SETUP

START

*S6,ACCDEM

*P302
20330       AOS CHARCOUNT(TAC1)      ;INCREMENT THE APPROPRIATE SLOT

*I.
20300       AOS CHARCOUNT(TAC)       ;INCREMENT THE APPROPRIATE SLOT
*ILR=
*P320,346
20320     CALL EXIT      ;UNLESS WE HAVE ALREADY READ 5000
20330
20340     RESET: SIXBIT /RESET/

*E

*^C
OPR
FINISHED WITH DRIVE 6 TKS GA
OK BYE
^C
KJOB
```